Orbits of monomials and factorization into products of linear forms

Pascal Koiran, Nicolas Ressayre

Orbits of monomials and factorization into products of linear forms*

Pascal Koiran
Université de Lyon, Ecole Normale Supérieure de Lyon, LIP†
Nicolas Ressayre
Univ Lyon, Université Claude Bernard Lyon 1,
Institut Camille Jordan (CNRS UMR 5208),
43 blvd. du 11 novembre 1918, F-69622 Villeurbanne cedex, France

July 10, 2018

Abstract

This paper is devoted to the factorization of multivariate polynomials into products of linear forms, a problem which has applications to differential algebra, to the resolution of systems of polynomial equations and to Waring decomposition (i.e., decomposition in sums of d-th powers of linear forms; this problem is also known as *symmetric tensor decomposition*). We provide three black box algorithms for this problem.

Our main contribution is an algorithm motivated by the application to Waring decomposition. This algorithm reduces the corresponding factorization problem to simultaenous matrix diagonalization, a standard task in linear algebra. The algorithm relies on ideas from invariant theory, and more specifically on Lie algebras.

Our second algorithm reconstructs a factorization from several bivariate projections. Our third algorithm reconstructs it from the determination of the zero set of the input polynomial, which is a union of hyperplanes.

1 Introduction

The main contribution of this paper is a simple algorithm which determines whether an input polynomial $f(x_1, \ldots, x_n)$ has a factorization of the form

$$f(x) = l_1(x)^{\alpha_1} \cdots l_n(x)^{\alpha_n} \tag{1}$$

*The authors are supported by ANR project CompA (code ANR–13–BS02–0001–01).
Email: pascal.koiran@ens-lyon.fr, ressayre@math.univ-lyon1.fr
†UMR 5668 ENS Lyon, CNRS, UCBL.

where the linear forms l_i are linearly independent. The algorithm outputs such a factorization if there is one. Our algorithm works in the black box model: we assume that we have access to the input polynomial f only through a "black box" which on input $(x_1, \ldots, x_n)$ outputs $f(x_1, \ldots, x_n)$.

We therefore deal with a very special case of the polynomial factorization problem. As explained in Section 1.2 below, this special case already has an interesting application to Waring decomposition. The algorithm is based on (elementary) ideas of invariant theory, but is nonetheless quite simple: it essentially boils down to the simultaneous diagonalization of commuting matrices, a standard task in linear algebra. For the general problem of factorization in the black box model there is a rather involved algorithm by Kaltofen and Trager [21], see Section 1.3 for more details. Our factorization algorithm seems to be the first to rely on ideas from invariant theory, and to reduce a multivariate polynomial factorization problem to matrix diagonalization. Let us now explain why it is natural to use invariant theory in this context.

1.1 Connection with invariant theory

Consider a field K of characteristic 0 and a polynomial $f \in K[x_1, \ldots, x_n]$. By definition, the orbit $\mathrm{Orb}(f)$ of f under the action of the general linear group is the set of polynomials of the form $f(A.x)$ where $A \in GL_n(K)$ is an arbitrary invertible matrix. In their Geometric Complexity Theory program [32, 33], Mulmuley and Sohoni have proposed the following approach to lower bounds in algebraic complexity: in order to prove a lower bound for a polynomial g, show that it does not belong to a suitable *orbit closure* $\overline{\mathrm{Orb}(f)}$. The case where f is the determinant polynomial is of particular interest as it allows to address the infamous "permanent versus determinant" problem. Mulmuley and Sohoni have also proposed a specific representation-theoretic approach to deal with this orbit closure problem. As it turns out, the representation-theoretic approach provably does not work [8]. The general approach based on orbit closure remains plausible, but has so far not produced any major lower bound result because the orbit closure of the determinant is difficult to describe. By contrast, the renewed interest in invariant theory has led to new positive results, i.e., to new polynomial time algorithms: see for instance [7, 6, 18, 31] and especially [25], which is a main inspiration for this paper.

We deal here with the simplest of all orbits, namely, the orbit of a single monomial $x_1^{\alpha_1} \ldots x_n^{\alpha_n}$, and we derive a new factorization algorithm. It is immediate from the definition that this orbit is the set of polynomials that can be factorized as in (1) with linearly independent forms. Note that the orbit closure of the monomial $x_1 x_2 \ldots x_n$ is the set of polynomials that can be written as products of n linear forms (without any assumption of linear independence). This is well known in algebraic geometry, see example (5) in

Section 3.1.2 of [27] and exercise 3.1.4.2 in the same book. Moreover, equations for this orbit closure are known, see chapter 9 of [27] for a derivation of the equations and the history of this subject. However, no factorization algorithm relying on ideas from invariant theory is currently known for arbitrary products of linear forms. We suggest this problem as a natural step before considering more complicated orbit closure problems.

1.2 Application to Waring decomposition

The factorization problem studied here is motivated mainly by an algorithm due to Neeraj Kayal (see Section 5 of [24]). Factorization in products of linear forms is also useful for algorithmic differential algebra [36, 38] and for the resolution of systems of algebraic equations by factorization of the U-resultant [14, 26].

Kayal's algorithm determines whether a homogeneous polynomial f of degree d in n variables can be written as the sum of n d-th powers of linearly independent forms. This algorithm is based on the fact that such a polynomial has a Hessian determinant which factors as a product of $(d-2)$-th powers of linearly independent forms. In [24] the Hessian is factorized with Kaltofen's algorithm for the factorization of polynomials given by straight-line programs [23]. The decomposition of f as a sum of d-th powers can be recovered from this information. The algorithm presented in this note can therefore be used instead of Kaltofen's algorithm to solve the same decomposition problem.

Building on these ideas from [24], it was recently shown in [17] how to recover up to $O(n^2)$ terms in a Waring decomposition[1] (and more generally in a sum of powers of affine forms with possibly different exponents in each power). The algorithm works for polynomials of degree $d \geq 5$ and is based on the factorization of a "generalized Hessian" into products of linear forms. There are now up to order n^2 distinct linear forms in the factorization, and that many linear forms must of course be linearly dependent. This provides further motivation for the problem suggested at the end of Section 1.1 (namely, the extension of our algorithm to the case of linearly dependent forms). Factorization in products of dependent forms is discussed at the end of Section 1.3.

1.3 Comparison with previous factorization algorithms

As mentioned above, the algorithm for Waring decomposition in [24] relies on Kaltofen's factorization algorithm [23] which works in the arithmetic circuit (or "straight-line program") model: the input polynomial f is described by

[1] This algorithm works when the linear forms to be recovered are sufficiently generic; efficient reconstruction in the worst case is still open.

an arithmetic circuit, and the output is a list of arithmetic circuits for the irreducible factors of f together with their multiplicities.

One could instead appeal to the black-box factorization algorithm by Kaltofen and Trager [21]. In this case, instead of factorizing a circuit for the determinant of a Hessian matrix one would use a black box for the determinant of this matrix. The algorithm from [21] produces a black box for the irreducible factors of f given a black-box for evaluating f.

Compared to [21, 23] our algorithm works in a hybrid model: we use the most general of the two for the input polynomial (black box representation) but we explicitly determine the linear forms l_i in (1) when they exist.[2] For the general polynomial factorization problem, it is apparently not known how to efficiently produce "small" arithmetic circuits for the irreducible factors of a polynomial f given a black-box for f. Due to the black box algorithm of [21], this would be equivalent to producing a small arithmetic circuit for a polynomial given a black box for this polynomial.

The algorithms from [21, 23] project the original n-variate factorization problem to a bivariate factorization problem, solve the bivariate problem using a factorization algorithm for polynomials in dense representation, and then lift the result to a factorization of the n-variate input polynomial. It will be clear that our algorithm is based on a very different principle: instead of projecting we do linear algebra computations directly in n-dimensional space.

There is an intringuing connection between our algorithm and Gao's algorithm for the absolute factorization of bivariate polynomials [16]: they are both based on the study of certains partial differential equations. For the connection of our approach to PDEs see Lemma 5 in Section 2.3.

As explained in Section 1.2, for the application to Waring decomposition following [24] we can assume that the linear forms l_i are independent. This assumption does not seem so natural in other applications such as differential algebra [36, 38] or the resolution of systems of polynomial equations [14, 26]. For this reason, we present in Section 5 another algorithm for factorization into products of linear forms based like [21, 23] on bivariate projections. Our goal in that section is to give a simpler algorithm which takes advantage of the fact that we are considering only a special case of the polynomial factorization problem. We present another simple algorithm in Section 6. This algorithm requires a univariate factorization algorithm, and the projection-based algorithm requires a bivariate factorization algorithm (see Sections 5 and 6 for more details).

For these last two algorithms, no assumption of linear independence is needed. This is also the case for the algorithms in [26, 38]. In these two papers no complexity analysis is provided, and it is assumed in the second

[2]It would anyway be easy to explicitly determine the l_i by interpolation from a black box for these linear forms.

one that the polynomial to be factorized is squarefree. We note that the algorithm from [26] bears some similarity to the algorithm that we present in Section 6: both are based on the determination of the zero set of the input polynomial, which is a union of hyperplanes.

1.4 On the choice of fields

Polynomial factorization problems come with many variations. In particular, the following choices need to be made:

(i) The input is a polynomial $f \in K[x_1, \ldots, x_n]$. What field K do we choose as field of coefficients for f?

(ii) What field $\mathbb{K}$ do we choose as field of coefficients for the output? More precisely, the output is a factorization $f = g_1 \ldots g_k$ where the polynomials g_i belong to $\mathbb{K}[x_1, \ldots, x_n]$ for some field extension $\mathbb{K}$ of K, and are irreducible over $\mathbb{K}$. In the literature it is often (but not always) assumed that $K = \mathbb{K}$.

(iii) How do we represent the field elements? Assume for instance that $K = \mathbb{Q}$ and that we are interested in absolute factorization, i.e., factorization over $\mathbb{K} = \overline{\mathbb{Q}}$ (the algebraic closure of $\mathbb{Q}$). Do we insist on a symbolic representation for the coefficients of the g_i's (in this case, the coefficients would be represented as elements of an extension of $\mathbb{Q}$ of finite degree) or, using an embedding $\overline{\mathbb{Q}} \subseteq \mathbb{C}$, are we happy to compute only numerical approximations of these coefficients?

Absolute factorization seems to be the most natural choice for this paper because of the application to Waring decomposition (this problem has been studied mostly in algebraically closed fields[3]). Moreover, for any field $\mathbb{K}$ if a decomposition of f of the form (1) with the l_i in $\mathbb{K}[x_1, \ldots, x_n]$ is possible then this decomposition clearly is an absolute factorization of f.

Nevertheless, we do not commit to any specific choice for (i), (ii) and (iii) except that K must be of characteristic zero. This is possible because our main algorithm is a *reduction* (to matrix diagonalization). Any (efficient) algorithm for this standard linear algebra task for a specific choice of (i), (ii) and (iii) will therefore yield an (efficient) factorization algorithm. We elaborate on the complexity of our reduction in Section 1.5.

1.5 Complexity of our invariant-theoretic algorithm

The black box algorithm in [21] applies to polynomials with coefficients in a field K of characteristic 0. The only assumption on K if that a factorization algorithm for univariate polynomials in $K[x]$ is available. This black

[3]Some results are also known for the field of real numbers [11, 13]

box algorithm can therefore be thought of as a reduction from multivariate to univariate polynomial factorization. In order to evaluate precisely the complexity of this algorithm for a specific field K, one must of course take into account the complexity of the univariate factorization problem for this particular field.

Likewise, our main algorithm can be thought of as a reduction to (simultaneous) matrix diagonalization.[4] When we write that the algorithms of Section 4 run in polynomial time, we mean polynomial in n (the number of variables of the input polynomial) and d (its degree). In particular, the algorithm makes $\mathrm{poly}(n, d)$ calls to the black box for f. It also performs simultaneous diagonalization on n (commuting) matrices, and makes a few other auxiliary computations. The main one is the determination of the Lie algebra of f, which as explained in Section 2.3 is a linear algebra problem; a polynomial black box algorithm for it can be found in [25]. A more precise analysis of our algorithm can be found in the appendix. It suggests that the computation of the Lie algebra of f is a particularly expensive step. Improving the algorithm from [25] (or its analysis in the appendix) seems to be an interesting open problem.

If we just want to *decide* the existence of a suitable factorization (rather than compute it) our algorithm becomes purely algebraic, i.e., it just performs arithmetic operations (additions, multiplications and tests to zero) on the function values given by the black box for f. In particular, we do not need to factor univariate polynomials or diagonalize matrices.

Like in [23, 21] our algorithm is randomized and can return a wrong answer with a small probability ϵ. This is unavoidable because homogeneous polynomials of degree d in n variables have $\binom{n+d-1}{d}$ coefficients and this is bigger than any fixed polynomial in n and d if these two parameters are nonconstant. As a result, for a polynomial f of form (1) there will always be another polynomial g which agrees with f on all points queried on input f. The algorithm will therefore erroneously[5] output the same answer on these two inputs. The probability of error ϵ can be thought of as a small fixed constant, and as usual it can be made as small as desired by repeating the algorithm (or by changing the parameters in the algorithm from [25] for the computation of the Lie algebra; this is the main source of randomness in our algorithm[6]).

[4] Note that diagonalizing a matrix is clearly related to the factorization of its characteristic polynomial.

[5] Indeed, the algorithm should report failure if g is not of form (1), or if it is should return a different factorization than for f.

[6] If f is given explicitly as a sum of monomials, the Lie algebra can be computed deterministically in polynomial time; this is clear from the characterization of the Lie algebra in Lemma 5.

1.6 Organization of the paper

In Section 2 we recall some background on matrix diagonalization, simultaenous diagonalization and invariant theory. In Section 3 we give a characterization of the polynomials in the orbit of a monomial. We use this characterization in Section 4 to derive our main algorithm for factorization into products of (independent) linear forms. An algorithm based on the older idea of bivariate projections is presented in Section 5. In contrast to [23, 21] this algorithm recovers a factorization of the input polynomial from *several* bivariate projections. Another simple algorithm is presented in Section 6. As mentioned earlier, this algorithm relies on the determination of the zero set of f. Our last two algorithms do not rely on any invariant theory and do not require any independence property for the linear forms. As pointed out at the end of Section 1.1, for factorization into products of arbitrary linear forms no algorithm that would rely on ideas from invariant theory is known at this time.

The paper ends with two appendices where we analyze the complexity of our three algorithms in more detail than in the main body. In particular, we point out in Appendix A an optimization of our invariant-theoretic algorithm for the "white box" model, in which the black box for f is implemented by an arithmetic circuit.

2 Background

We first recall the Schwarz-Zippel lemma [35, 42], a ubiquitous tool in the analysis of randomized algorithms.

Lemma 1. *Let $f \in K[x_1, \ldots, x_n]$ be a nonzero polynomial. If $a_1, \ldots, a_n$ are drawn independently and uniformly at random from a finite set $S \subseteq K$ then*

$$\Pr[f(a_1, \ldots, a_n) = 0] \leq \deg(f)/|S|.$$

A similar result with a slightly worse bound was obtained a little earlier by DeMillo and Lipton [15]. In the remainder of this section we recall some background on matrix diagonalization, on simultaenous diagonalization, on invariant theory and Lie algebras.

2.1 Background on matrix diagonalization

Since our main algorithm is a reduction to matrix diagonalization, it is appropriate to provide some brief background on the algorithmic solutions to this classical problem. After a first course on linear algebra, this might look like a simple task: to diagonalize a matrix M, first compute its eigenvalues. Then, for each eigenvalue λ compute a basis of $\mathrm{Ker}(M - \lambda.I)$. But this problem is more subtle than it seems at first sight.

Let us begin with numerical algorithms. There is a vast literature on numerical methods for eigenvalue problems (see for instance [2] and the references there). Naively, one might want to compute the eigenvalues of M by computing the roots of its characteristic polynomial $\chi_M(\lambda) = \det(M - \lambda I)$. This approach is hardly ever used in practice for large matrices because the roots of a polynomial can be very sensitive to perturbations of its coefficients [41]. A theoretical analysis explaining why such a bad behaviour is rather prevalent can be found in [9]. The QR algorithm is now considered to be the standard algorithm for computing all eigenvalues and eigenvectors of a dense matrix [2]. It works well in practice, but a thorough understanding of this algorithm (or of *any* efficient and stable numerical algorithm for the computation of eigenvalue – eigenvector pairs) is still lacking, see Open Problem 2 in [5].

Let us now turn to symbolic methods. In the absence of roundoff errors, an approach based on the computation of the characteristic polynomial becomes feasible (see [34] for the state of the art on the computation of this polynomial). From the knowledge of χ_M we can decide whether M is diagonalizable using the following classical result from linear algebra.

Proposition 2. *Let K be a field of characteristic 0 and let χ_M be the characteristic polynomial of a matrix $M \in M_n(K)$. Let $P_M = \chi_M / \gcd(\chi_M, \chi'_M)$ be the squarefree part of χ_M. The matrix M is diagonalizable over $\overline{K}$ iff $P_M(M) = 0$.*[7] *Moreover, in this case M is diagonalizable over K iff all the roots of P_M lie in K.*

Once we know that M is diagonalizable, computing the diagonal form of M symbolically requires the factorization of P_M. We note that for $K = \mathbb{Q}$, finding the roots of P_M in $\mathbb{Q}$ is cheaper than the general problem of factorization in irreducible factors over $\mathbb{Q}[X]$ ([3], Proposition 21.22). This faster algorithm should therefore be used to diagonalize over $\mathbb{Q}$. For the purpose of this paper, this is relevant for factorisation into a product of linear forms with rational coefficients.

Once we know the eigenvalues of M and their multiplicities, the last step is the computation of a transition matrix T such that $T^{-1}MT$ is diagonal. For this step we refer to [19, 20, 39]. These papers consider the more general problem of computing symbolic representations of the Jordan normal form.

The knowledge of T is particularly important for the application to factorization into product of linear forms because (as shown in Section 4) these forms can be read off directly from the transition matrix. If we just want to know whether such a factorization is possible over K or $\overline{K}$, Proposition 2 is sufficient.

[7] An equivalent characterization is that the minimal polynomial of M has only simple roots.

2.2 Simultaneous diagonalization

It is a well known fact of linear algebra that a family of diagonalizable matrices is simultaneously diagonalizable if and only if they pairwise commute. We will use this criterion to test whether a family of matrices $A_1, \ldots, A_k$ is simultaneously diagonalizable. If the test succeeds, we will then need to diagonalize them. Note that a transition matrix which diagonalizes A_1 may not necessarily diagonalize the other matrices (this may happen if A_1 has an eigenvalue of multiplicity larger than 1). We can nonetheless perform a simultaneous diagonalization by diagonalizing a single matrix. Indeed, as suggested in Section 6.1.1 of [25] we can diagonalize a random linear combination of the A_i's. We sketch a proof of this simple fact below. For notational simplicity we consider only the case of two matrices. The general case can be treated in a similar way.

Lemma 3. *Assume that $M, N \in M_n(k)$ are two simultaneously diagonalizable matrices. There is a set $B \subseteq K$ of size at most $n(n-1)/2$ such that for any $t \in K \setminus B$ any eigenvector of $M + tN$ is also an eigenvector of M and N.*

Proof. Since M and N are simultaneously diagonalizable we may as well work in a basis where these matrices become diagonal. We therefore assume without loss of generality that $M = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ and $N = \mathrm{diag}(\mu_1, \ldots, \mu_n)$. We then have $M + tN = \mathrm{diag}(\lambda_1 + t\mu_1, \ldots, \lambda_n + t\mu_n)$ for any $t \in K$. We may take for B the set of t's such that $\lambda_i + t\mu_i = \lambda_j + t\mu_j$ for some pair $\{i, j\}$ such that $(\lambda_i, \mu_i) \neq (\lambda_j, \mu_j)$. This is indeed a set of size at most $n(n-1)/2$, and for $t \notin B$ the eigenspace of $M + tN$ associated to the eigenvalue $\lambda_i + t\mu_i$ is the intersection of the eigenspace of M associated to λ_i and of the eigenspace of N associated to μ_i. In particular, any eigenvector of $M + tN$ is also an eigenvector of M and N. □

Proposition 4. *Assume that $M, N \in M_n(k)$ are two simultaneously diagonalizable matrices and that t is drawn from the uniform distribution on a finite set $S \subset K$. With probability at least $1 - \frac{n(n-1)}{2|S|}$, all the transition matrices which diagonalize $M + tN$ also diagonalize M and N.*

Proof. We show that the required property holds true for any t that does not belong to the "bad set" of Lemma 3.

For an invertible matrix T, $T^{-1}(M + tN)T$ is diagonal iff all the column vectors of T are eigenvectors of $M + tN$. But for $t \notin B$, any eigenvector of $M + tN$ is also an eigenvector of M and N. As a result, if $T^{-1}(M + tN)T$ is diagonal then $T^{-1}MT$ and $T^{-1}NT$ are diagonal as well. □

2.3 Background on invariants and Lie algebras

In this section and in the remainder of the paper, K denotes a field of characteristic 0. The general linear group GL_n acts on the polynomial ring

$K[x_1, \ldots, x_n]$ by linear change of variables: an invertible matrix $A \in GL_n$ sends a polynomial $P(x) \in K[x_1, \ldots, x_n]$ to $P(A.x)$. The group of invariant of P is the group of matrices A such that $P(A.x) = P(x)$. We recall that this is a Lie group. Its Lie algebra $\mathfrak{g}$ is a linear subspace of $M_n(K)$ defined as the tangent space of G at identity. More precisely, $\mathfrak{g}$ is the "linear part" of the tangent space; the (affine) tangent space is $I + \mathfrak{g}$.

The Lie algebra associated to the group of invariants of P will be called simply "Lie algebra of P", and we will denote it by $\mathfrak{g}_P$. It can be explicitly computed as follows.

Lemma 5 (Claim 59 in [25])**.** *A matrix $A = (a_{ij}) \in M_n(K)$ belongs to the Lie algebra of P if and only if*

$$\sum_{i,j \in [n]} a_{ij} x_j \frac{\partial P}{\partial x_i} = 0 \tag{2}$$

The elements of the Lie algebra therefore correspond to linear dependence relations between the polynomials $x_j \frac{\partial P}{\partial x_i}$.

As an example we determine the group of invariants of monomials.

Lemma 6. *The group of invariants of a monomial $m = x_1^{\alpha_1} \ldots . x_n^{\alpha_n}$ with $\alpha_i \geq 1$ for all i is generated by:*

(i) The diagonal matrices $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ *with $\prod_{i=1}^n \lambda_i^{\alpha_i} = 1$. We denote this subgroup of GL_n by T_α, where α is the tuple $(\alpha_1, \ldots, \alpha_n)$.*

(ii) The permutation matrices which map any variable x_i to a variable $x_{\pi(i)}$ with same exponent in m (i.e., with $\alpha_i = \alpha_{\pi(i)}$).

Proof. The monomial is obviously invariant under the actions of matrices from (i) and (ii). Conversely, assume that m is invariant under the action of an invertible matrix A. By uniqueness of factorization, A must send every variable x_i to the multiple of another variable, i.e., to $\lambda_i x_{\pi(i)}$. Moreover we must have $\alpha_i = \alpha_{\pi(i)}$ and $\prod_{i=1}^n \lambda_i^{\alpha_i} = 1$, so A is in the group generated by (i) and (ii). □

The Lie algebras of monomials is determined in Proposition 8. In this paper we will follow the Lie-algebraic approach from [25]. As a result we will not work directly with groups of invariants.

If two polynomials are equivalent under the action of GL_n, their Lie algebras are conjugate. More precisely:

Proposition 7 (Proposition 58 in [25])**.** *If $P(x) = Q(A.x)$ then*

$$\mathfrak{g}_P = A^{-1}.\mathfrak{g}_Q.A$$

3 The orbit of a monomial

Throughout the paper, m denotes a monomial $x_1^{\alpha_1}....x_n^{\alpha_n}$ with all exponents $\alpha_i \geq 1$.

Proposition 8. *The Lie algebra $\mathfrak{g}_m$ of a monomial $m = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ with all exponents $\alpha_i \geq 1$ is the space of diagonal matrices $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ such that $\sum_{i=1}^n \alpha_i \lambda_i = 0$.*

Proof. By Lemma 5, all these matrices are in $\mathfrak{g}_m$ since m satisfies the equation $x_i \frac{\partial m}{\partial x_i} = \alpha_i m$. Conversely, if $A \in \mathfrak{g}$ all off-diagonal entries a_{ij} must vanish since the monomial $x_j \frac{\partial m}{\partial x_i}$ could not cancel with any other monomial in (2). □

Remark 9. *The above characterization of $\mathfrak{g}_m$ is no longer true if some exponents α_i may vanish. Indeed, in this case there is no constraint on the entries in row i of a matrix in $\mathfrak{g}_m$. However, we note for later use that in all cases, the space of diagonal matrices $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ which lie in $\mathfrak{g}_m$ is defined by $\sum_{i=1}^n \alpha_i \lambda_i = 0$.*

It is easy to check by a direct computation that the Lie algebra determined in Proposition 8 is (as expected) equal to the tangent space at identity of the group T_α from Lemma 6. The next result turns Proposition 8 into an equivalence.

Proposition 10. *Let $f \in K[x_1, \ldots, x_n]$ be a homogeneous polynomial of degree d. The two following properties are equivalent:*

(i) f is a monomial which depends on all of its n variables.

(ii) The Lie algebra of f is an $(n-1)$-dimensional subspace of the space of diagonal matrices.

Proof. We have seen in Proposition 8 that (i) implies (ii). Conversely, for any polynomial P let us denote by $\mathfrak{d}_P$ the subspace of its Lie algebra made of diagonal matrices. By Lemma 5, $\mathfrak{d}_f$ is the space of of matrices $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ such that

$$\sum_{i=1}^n \lambda_i x_i \frac{\partial f}{\partial x_i} = 0 \tag{3}$$

For any monomial m, $x_i \frac{\partial m}{\partial x_i}$ is proportional to m. This implies that $\mathfrak{d}_f$ is the intersection of the $\mathfrak{d}_m$'s for the various monomials m appearing in f since the contributions to (3) coming from different monomials cannot cancel. By Remark 9, for two distinct monomials m_1 and m_2 appearing in f the subspaces $\mathfrak{d}_{m_1}$ and $\mathfrak{d}_{m_2}$ are distinct since they are defined by linear forms that are not proportional (here we use the homogeneity of f). It follows that their intersection is of dimension $n-2$ in contradiction with (ii). Therefore,

only one monomial can appear in f. Finally, by Remark 9 all of the n variables must appear in this monomial; otherwise, $\mathfrak{g}_f$ would contain some nondiagonal matrices. □

We can now characterize the Lie algebras of polynomials in the orbit of a monomial.

Theorem 11. *Consider a monomial $m = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ with $\alpha_i \geq 1$ for all i, a homogeneous polynomial $f \in K[x_1, \ldots, x_n]$ of degree $d = \alpha_1 + \cdots + \alpha_n$ and an invertible matrix A. The two following properties are equivalent.*

(i) The action of A sends m to a multiple of f, i.e., $m(A.x) = c.f(x)$ for some constant c.

(ii) The Lie algebras of f and m are conjugate by A, i.e., $\mathfrak{g}_f = A^{-1}.\mathfrak{g}_m.A$.

Proof. Proposition 7 shows that (i) implies (ii). For the converse, assume that $\mathfrak{g}_f = A^{-1}.\mathfrak{g}_m.A$ and define $g(x) = f(A^{-1}.x)$. By Proposition 7 we have $\mathfrak{g}_g = \mathfrak{g}_m$. It follows from Propositions 8 and 10 that $g = \lambda.m$ for some nonzero constant λ. We therefore have $m(Ax) = f(x)/\lambda$. □

This characterization takes a particularly simple form in the case of equal exponents.

Theorem 12. *Consider a monomial $m = (x_1 \cdots x_n)^\alpha$ and a homogeneous polynomial $f \in K[x_1, \ldots, x_n]$ of degree $d = n\alpha$. The two following properties are equivalent.*

(i) Some multiple of f belongs to the orbit of m, i.e., $m(A.x) = c.f(x)$ for some invertible matrix A and some constant c.

(ii) The Lie algebra of f has a basis made of $n-1$ diagonalizable matrices of trace zero which pairwise commute.

Moreover, f is a constant multiple of m if and only its Lie algebra is the space of diagonal matrices of trace zero.

Proof. Let f be in the orbit of m. By Proposition 7, in order to establish (ii) for f we just need to check that this property is true for m. This is the case since (by Proposition 8) the Lie algebra of m is the space of diagonal matrices of trace 0.

Conversely, assume that (ii) holds for f. It is a well known fact of linear algebra that a family of diagonalizable matrices is simultaneously diagonalizable if and only if they pairwise commute. By simultaneously diagonalizing the $n-1$ matrices in the basis of $\mathfrak{g}_f$ we find that this Lie algebra is conjugate to $\mathfrak{g}_m$ (which as we just saw is the space of diagonal matrices of trace 0). Hence some constant multiple of f is in the orbit of m by Theorem 11.

As to the second part of the theorem, we have already seen that $\mathfrak{g}_m$ is the space of diagonal matrices of trace zero. Conversely, if $\mathfrak{g}_f = \mathfrak{g}_m$ we can apply Theorem 11 with $A = \mathrm{Id}$ and it follows that f is a constant multiple of m. □

Note that if (ii) holds for some basis of $\mathfrak{g}_f$ this property holds for all bases. Also, if K is algebraically closed we can always take $c = 1$ in Theorems 11 and 12.

4 Factorization into products of independent forms

By definition, the orbit of a monomial $m = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ contains the polynomial f if an only if f can be written as $f(x) = l_1(x)^{\alpha_1} \cdots l_n(x)^{\alpha_n}$ where the linear forms l_i are linearly independent. We will exploit the characterization of orbits obtained in Section 3 to factor such polynomials. We assume that we have access to a black-box for f. We begin with the simpler case of equal exponents. Note that this is exactly what is needed in Section 5 of [24].

4.1 Equal exponents

In this section we describe an algorithm which takes as input a homogeneous polynomial $f \in K[x_1, \ldots, x_n]$ of degree $d = n\alpha$, determines if it can be expressed as $f = (l_1 \cdots l_n)^\alpha$ where the l_i's are linearly independent forms and finds such a factorization if it exists. In the first three steps of the following algorithm we decide whether such a factorization exists over $\overline{K}$, and in the last two we actually compute the factorization.

1. Compute a basis $B_1, \ldots, B_k$ of the Lie algebra of f.

2. Reject if $k \neq n - 1$, i.e., if the Lie algebra is not of dimension $n - 1$.

3. Check that the matrices $B_1, \ldots, B_{n-1}$ commute, are all diagonalizable over $\overline{K}$ and of trace zero. If this is the case, declare that f can be factored as $f = (l_1 \cdots l_n)^\alpha$ where the l_i's are linearly independent forms. Otherwise, reject.

4. Perform a simultaneous diagonalization of the B_i's, i.e., find an invertible matrix A such that the $n - 1$ matrices AB_iA^{-1} are diagonal.

5. At the previous step we have found a matrix A such that $f(A^{-1}x) = \lambda.m(x)$ where m is the monomial $(x_1 \cdots x_n)^\alpha$. We therefore have $f(x) = \lambda.m(Ax)$ and we output this factorization.

Note that this algorithm outputs a factorization of the form $f = \lambda.(l_1 \cdots l_n)^\alpha$. We can of course obtain $\lambda = 1$ by an appropriate scaling of the l_i's if desired.

Theorem 13. *The above algorithm runs in polynomial time and determines whether f can be written as $f = (l_1 \cdots l_n)^\alpha$ where the forms l_i are linearly independent. It ouputs such a factorization if there is one.*

Proof. The correctness of the algorithm follows from Theorem 12. In particular, the equivalence of properties (i) and (ii) in Theorem 12 shows that the algorithm will make a correct decision on the existence of a suitable factorization at step 3. If this step succeeds, the simultaneous diagonalization at step 4 is possible since (as already pointed out Section 2.2 and in the proof of Theorem 12) simultaneous diagonalization is always possible for a family of matrices which are diagonalizable and pairwise commute. By Proposition 7, the Lie algebra of $f(A^{-1}x)$ is the space of diagonal matrices of trace 0. This implies that $f(A^{-1}x)$ is a constant multiple of m by the last part of Theorem 12, and justifies the last step of the algorithm.

Let us now explain how to implement the 5 steps. A randomized[8] black box algorithm for Step 1 based on Lemma 5 can be found in Lemma 22 of [25]. Steps 2 and 3 are mostly routine (use Proposition 2 to check that the B_i's are diagonalizable). Step 4 (simultaenous diagonalization of commuting matrices) is also a standard linear alegbra computation. One suggestion from Section 6.1.1 of [25] is to diagonalize a random linear combination of the B_i's (see Section 2.2 for more details). That matrix can be diagonalized as explained in Section 2.1. Finally, the scaling factor λ at step 5 can be computed by one call to the black box for f. □

Remark 14. *We have presented the above algorithm with a view towards factorisation over $\overline{K}$, but it is readily adapted to factorization over some intermediate field $K \subseteq \mathbb{K} \subseteq \overline{K}$. Note in particular that to decide the existence of a factorization at step 3, we would need to check that the matrices B_i are diagonalizable over $\mathbb{K}$. As recalled in Proposition 2, this requires an algorithm that decides whether the characteristic polynomial of a matrix has all its roots in $\mathbb{K}$. In the case $\mathbb{K} = \overline{K}$, if we stop at step 3 we obtain a purely algebraic algorithm for deciding the existence of a suitable factorization (in particular, we do not need to factorize univariate polynomials or diagonalize matrices).*

4.2 General case

In this section we describe an algorithm which takes as input a homogeneous polynomial f of degree $d = \alpha_1 + \cdots + \alpha_n$ in n variables, determines if it can be expressed as $f(x) = l_1(x)^{\alpha_1} \cdots l_n(x)^{\alpha_n}$ where the l_i's are linearly independent forms, and finds such a factorization if it exists. Note that the values of the exponents α_i are determined by the algorithm (they are not given as input). We assume that $\alpha_i \geq 1$ for all i. The number of distinct factors is therefore

[8] There is no need for randomization if f is given explicitly as a sum of monomials rather than by a black box (in this case we can directly solve the linear system from Lemma 5).

equal to the number of variables of f. The case where there are more factors than variables is related to orbit closure and we do not treat it in this section. Let us explain briefly explain how to handle the case where some exponents α_i may be 0, i.e., the case where the number r of distinct factors is smaller than the number of variables. In this case, f has only r "essential variables", i.e., it is possible to make a linear (invertible) change of variables after which f depends only on r variables. This puts us therefore in the situation where the number of distinct factors is equal to the number of variables. The number of essential variables and the corresponding change of variables can be computed with Kayal's algorithm[9] [24], see also [10].

We can now present our factorization algorithm. Like in the case of equal exponents, the existence of a suitable factorization is decided in the first three steps.

1. Compute a basis $B_1, \ldots, B_k$ of the Lie algebra of f.
2. Reject if $k \neq n-1$, i.e., if the Lie algebra is not of dimension $n-1$.
3. Check that the matrices $B_1, \ldots, B_{n-1}$ commute and are all diagonalizable over $\overline{K}$. If this is not the case, reject. Otherwise, declare the existence of a factorization $f(x) = l_1(x)^{\alpha_1} \cdots l_n(x)^{\alpha_n}$ where the linear forms l_i are linearly independent and $\alpha_i \geq 1$ (the l_i and α_i will be determined in the last 3 steps of the algorithm).
4. Perform a simultaneous diagonalization of the B_i's, i.e., find an invertible matrix A such that the $n-1$ matrices AB_iA^{-1} are diagonal.
5. At the previous step we have found a matrix A such that $g(x) = f(A^{-1}x)$ has a Lie algebra $\mathfrak{g}_g$ which is an $(n-1)$-dimensional subspace of the space of diagonal matrices. Then we compute the orthogonal of $\mathfrak{g}_g$, i.e., we find a vector $\alpha = (\alpha_1, \ldots, \alpha_n)$ such $\mathfrak{g}_g$ is the space of matrices $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ satisfying $\sum_{i=1}^n \alpha_i \lambda_i = 0$. We normalize α so that $\sum_{i=1}^n \alpha_i = d$.
6. We must have $g(x) = \lambda.m$ where $\lambda \in K^*$ and m is the monomial $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ (in particular, α must be a vector with integral entries). We therefore have $f(x) = \lambda.m(Ax)$ and we output this factorization.

Again, this algorithm outputs a factorization of the form $f(x) = \lambda.l_1(x)^{\alpha_1} \cdots l_n(x)^{\alpha_n}$ and we can obtain $\lambda = 1$ by an appropriate scaling of the l_i's.

Theorem 15. *The above algorithm runs in polynomial time and determines whether f can be written as $f(x) = l_1(x)^{\alpha_1} \cdots l_n(x)^{\alpha_n}$ where the forms l_i are*

[9] The algorithm in [24] works in the circuit model, i.e., it is assumed that the input polynomial is given by an arithmetic circuit. Kayal later showed how to perform the same task in the black box model, see Section 3 of [25].

linearly independent and $\alpha_i \geq 1$ for all i. It ouputs such a factorization if there is one.

Proof. The two main steps (finding a basis of $\mathfrak{g}_f$ and simultaneous diagonalization) can be implemented efficiently as in the case of equal exponents, so we'll focus on the correctness of the algorithm.

Assume first that f can be written as $f(x) = L_1(x)^{\beta_1} \cdots L_n(x)^{\beta_n}$ where the L_i's are linearly independent forms and $\beta_i \geq 1$ for all i. Then f is in the orbit of the monomial $M = x_1^{\beta_1} \cdots x_n^{\beta_n}$, so $\mathfrak{g}_f$ and $\mathfrak{g}_M$ are conjugate by Proposition 7. By Proposition 8, $\mathfrak{g}_M$ is the space of diagonal matrices $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ such that $\sum_{i=1}^n \beta_i \lambda_i = 0$. These two facts imply that the first 4 steps of the algorithm will succeed. The polynomial $g(x) = f(A^{-1}x)$ defined at step 5 has a Lie algebra which is an $(n-1)$-dimensional subspace of the space of diagonal matrices. By Proposition 10, g must therefore be a monomial. Proposition 8 implies that the tuple of exponents of g is correctly determined at step 5, so that we indeed have $g = \lambda.m$ at step 6. Note that m may differ from M by a permutation of indices, and likewise the factorization output by the algorithm may differ from $f(x) = L_1(x)^{\beta_1} \cdots L_n(x)^{\beta_n}$ by a permutation of indices and the scaling of linear forms.

Conversely, if the 3 first steps of the algorithm succeed the B_i must be simultaneously diagonalizable and it follows again from Proposition 10 that the polynomial g defined at step 5 satisfies $g = \lambda.m$ where $\lambda \in K^*$ and m is some monomial $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. In particular, Proposition 10 guarantees that the exponents α_i are all positive. The algorithm will then output at step 6 a correct factorization of f. □

Like in Section 4.1 we have presented our algorithm with a view towards factorisation over $\overline{K}$, but it is readily adapted to factorization over some intermediate field $K \subseteq \mathbb{K} \subseteq \overline{K}$ as explained in Remark 14.

In the above algorithm we need to perform the simultaneous diagonalization at step 4 before computing the exponents α_i. In the remainder of this section we show that the exponents can be computed without step 4. The corresponding algorithm relies on Proposition 16 below. First, we recall that for any set of matrices $S \subseteq M_n(K)$ the centralizer of S is the set of matrices that commute with all matrices of S. It is a linear subspace of $M_n(K)$ and we denote it by $C(S)$.

Proposition 16. *Consider a monomial $m = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ with $\alpha_i \geq 1$ for all i, and a polynomial f in the orbit of m.*

The centralizer $C(\mathfrak{g}_f)$ of the Lie algebra of f is of dimension n. Moreover, there is a unique H in $C(\mathfrak{g}_f)$ such that $\mathrm{Tr}\, H = d$ and $\mathrm{Tr}(HM) = 0$ for all $M \in \mathfrak{g}_f$. The matrix H is diagonalizable, its eigenvalues are $(\alpha_1, \ldots, \alpha_n)$ and $C(\mathfrak{g}_f) = \mathfrak{g}_\mathfrak{f} \oplus \mathrm{Span}(H)$.

Note that the case $\alpha_1 = \ldots = \alpha_n = 1$ corresponds to $H = \mathrm{Id}$. The

condition $\mathrm{Tr}(HM) = 0$ for all $M \in \mathfrak{g}_f$ is an analogue of the trace zero condition in property (ii) of Theorem 12.

Proof. We first consider the case $f = m$. By Proposition 8, $\mathfrak{g}_m$ is the set of diagonal matrices $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ such that $\sum_i \alpha_i \lambda_i = 0$.

For $1 \leq i \neq j \leq n$, let $\mathcal{H}_{ij}$ denote the set of matrices $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ such that $\lambda_i = \lambda_j$. Consider the set $\mathfrak{h}$ of diagonal matrices. The hyperplane $\mathfrak{g}_m$ of $\mathfrak{h}$ is equal to no hyperplane of the form $\mathcal{H}_{ij}$. Since the field is infinite, $\mathfrak{g}_m$ is not contained in the union of the hyperplanes $\mathcal{H}_{ij}$. Then $\mathfrak{g}_m$ contains a matrix M_0 with pairwise distinct eigenvalues. Then $\mathfrak{h} \subseteq C(\mathfrak{g}_m) \subseteq C(M_0) \subseteq \mathfrak{h}$, and $C(\mathfrak{g}_m) = \mathfrak{h}$.

Set $H_0 = \mathrm{diag}(\alpha_1, \ldots, \alpha_n) \in \mathfrak{h}$. It is clear that $\mathrm{Tr}(H_0) = d$ and $\mathrm{Tr}(H_0 M) = 0$ for any $M \in \mathfrak{g}_m$. Conversely, let $H = \mathrm{diag}(\beta_1, \ldots, \beta_n) \in \mathfrak{h}$ and $M = \mathrm{diag}(\lambda_1, \ldots, \lambda_n) \in \mathfrak{g}_m$. Then $\mathrm{Tr}(HM) = \sum_i \beta_i \lambda_i$. Since $\mathfrak{g}_m$ is the hyperplane of $\mathfrak{h}$ defined by $\sum_i \alpha_i \lambda_i = 0$, $\mathrm{Tr}(HM) = 0$, for any $M \in \mathfrak{g}_m$ if and only if $(\beta_1, \ldots, \beta_n)$ is proportional to $(\alpha_1, \ldots, \alpha_n)$. If moreover $\mathrm{Tr}(H) = d$, we get $H = H_0$. This proves the unicity. Morever, $\mathrm{Tr}(H_0^2) = \sum_i \alpha_i^2 \neq 0$ and $H_0 \notin \mathfrak{g}_m$, since the field has characteristic zero. Then, since $\mathfrak{g}_m$ is an hyperplane of $\mathfrak{h}$, $\mathfrak{g}_m \oplus KH_0 = C(\mathfrak{g}_m) = \mathfrak{h}$.

Consider now a point f in the orbit of m. Let A be an invertible matrix such that $f = A.m = m \circ A^{-1}$. Then, by Proposition 7, $\mathfrak{g}_f = A\mathfrak{g}_m A^{-1}$ and $C(\mathfrak{g}_f) = AC(\mathfrak{g}_m)A^{-1}$. One easily checks that H satisfies the proposition for f if and only if $A^{-1}HA$ satisfies it for m. With the first part, this proves the existence and unicity of H. □

This proposition yields the following algorithm for the computation of the exponents $\alpha_1, \ldots, \alpha_n$. We assume that the first three steps of the algorithm of Theorem 15 have executed successfully.

(a) Set up and solve the linear system which expresses that $\mathrm{Tr}[H] = d$, $\mathrm{Tr}[HB_i] = 0$ and $HB_i = B_i H$ for all $i = 1, \ldots, n-1$. Here $(B_1, \ldots, B_{n-1})$ is the basis of $\mathfrak{g}_f$ computed at step 1 of the algorithm of Theorem 15. The system's unknowns are the n^2 entries of H.

(b) Compute the eigenvalues $\alpha_1, \ldots, \alpha_n$ of H.

Note that the system constructed at step (a) is overdetermined: it has $\Theta(n^3)$ equations but only n^2 unknowns. Proposition 16 guarantees that the system has a unique solution H, and that the eigenvalues of H are the exponents $\alpha_1, \ldots, \alpha_n$. We refer to Section 2.1 for the computation of eigenvalues at step (b).

5 Bivariate projections

In this section we present a probabilistic black box algorithm that finds a factorization into products of linear forms whenever this is possible, without any assumption of linear independence of the linear forms. As explained before this can be done with the algorithm by Kaltofen and Trager [21].

We assume that the input polynomial is in $K[x_1, \ldots, x_n]$ where K is infinite. In contrast to Section 4, we do not need to assume that K is of characteristic 0. The hypothesis that K is infinite is needed because the algorithm draws random elements from "large enough" but finite subsets $S \subseteq K$. The algorithm also applies to a finite field if K is large enough for this, or if we can draw points from a large enough field extension.

As in [23, 21] we rely on bivariate projections but we present a simplified algorithm which takes advantage of the fact that we are trying to factor polynomials of a special form (another simple algorithm based on a different idea is presented in the next section). In these two papers, a factorization of the input polynomial is recovered from a single bivariate projection (see Step R in [23] and Step 1 in [21] [10]). By contrast, we will recover the solution to our problem from several projections as in e.g. [17, 25]. A recurring difficulty with projection-based algorithms is that when we try to "lift" the solutions of problems on a lower-dimensional space to a solution of the original problem, the lift may not be unique. We first present in Section 5.1 a solution under an additional assumption which guarantees uniqueness of the lift. We then lift (as it were) this assumption in Section 5.2.

We assume that a polynomial time factorization algorithm for polynomials in $K[x, y]$ is available. It is explained in [22] how to obtain such an algorithm from a univariate factorization algorithm for the field of rational numbers, and more generally for number fields and finite fields. In the case of absolute factorization, polynomial time algoritms were first given by Gao [16] and by Chèze and Lecerf [12]. The complexity of the latter algorithm was analyzed in [12] for the algebraic (unit cost) model of computation. The complexity of the former algorithm was analyzed in [16] for an input polynomial with coefficients in a finite field F_q.[11]

Without loss of generality, we'll assume that our input f is in $K[x_1, \ldots, x_n]$ with $n \geq 4$. Indeed, if there are only 3 variables we can set $g(x_1, x_2) = f(x_1, x_2, 1)$, use the bivariate algorithm to factor g as a product of affine forms, and homogonenize the result to obtain a factorization of f. Note that the homogenization step includes a multiplication by $x_3^{\deg(f)-\deg(g)}$.

[10] More precisely, the construction of the black boxes for the irreducible factors of f requires a single projection. Evaluating these black boxes at an input point requires another bivariate projection, see Step A in [21]

[11] The algorithm also works for fields of characteristic 0, but a precise analysis of its complexity was left for future work.

5.1 A uniqueness condition

In this section we assume that our input $f(x_1, \ldots, x_n)$ can be factorized as

$$f(x) = \lambda . l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k} \tag{4}$$

where the linear form l_i is not proportional to l_j if $i \neq j$, and λ is a nonzero constant. We would like to recover λ, the exponents α_i's and the l_i's (note that each linear forms is defined only up to a constant).

Write $l_i(x) = \sum_{j=1}^{n} l_{ij} x_j$. In order to guarantee "uniqueness of the lift" we make the following temporary assumption:

(*) The k coefficients l_{i1} are distinct and nonzero and $l_{in} = 1$ for all i.

The algorithm is as follows.

1. For $j = 2, \ldots, n-1$ define $g_j(x_1, x_j) = f \circ \pi_j$ where the projection π_j sends variable x_n to the constant 1, leaves x_1 and x_j unchanged and sets all other variables to 0. Compute the dense representation of the g_j's by interpolation.

2. Using the bivariate factorization algorithm, write each g_j as $g_j(x_1, x_j) = \lambda . \prod_{i=1}^{k} (a_{ij} x_1 + b_{ij} x_j + 1)^{\beta_{ij}}$.

3. At the beginning of this step, each of the $n-2$ tuples $(a_{1j}, \ldots, a_{kj})$ is a permutation of the tuple $(l_{11}, \ldots, l_{k1})$. We reorder the factors in the factorizations of the g_j from step 2 to make sure that the $n-2$ tuples are identical (i.e., its elements always appear in the same order). After reordering, the $n-2$ tuples of exponents $(\beta_{1j}, \ldots, \beta_{kj})$ will also become identical. We therefore obtain factorizations of the form:

$$g_j(x_1, x_j) = \lambda . \prod_{i=1}^{k} (a_i x_1 + c_{ij} x_j + 1)^{\gamma_i}.$$

4. We output the factorization:

$$f(x_1, \ldots, x_n) = \lambda . \prod_{i=1}^{k} (a_i x_1 + c_{i2} x_2 + \cdots + c_{i,n-1} x_{n-1} + x_n)^{\gamma_i}.$$

The main issue regarding the correctness of this algorithm is to make sure that we have correctly combined the factors of the g_j's to obtain the factors of f. This is established in the next proposition. For an example of what can go wrong without assumption (*) consider the following two polynomials:

$$f_1 = (x_1 + x_2 + x_3 + x_4)(x_1 + 2x_2 + 2x_3 + x_4)$$

and

$$f_2 = (x_1 + x_2 + 2x_3 + x_4)(x_1 + 2x_2 + x_3 + x_4).$$

At step 1 of the algorithm, these two polynomials are mapped to the same pair of bivariate polynomials:

$$g_2 = (x_1 + x_2 + 1)(x_1 + 2x_2 + 1),\ g_3 = (x_1 + x_3 + 1)(x_1 + 2x_3 + 1)$$

and there is no unique way of lifting $\{g_2, g_3\}$ to an input polynomial. Another difficulty is that the factorization pattern of f (i.e., the set of exponents $\{\alpha_1, \ldots, \alpha_k\}$) could change after projection, for instance

$$f = (x_1 + x_2 + x_3 + x_4)(x_1 + 2x_2 + x_3 + x_4)$$

is mapped to

$$g_2 = (x_1 + x_2 + 1)(x_1 + 2x_2 + 1),\ g_3 = (x_1 + x_3 + 1)^2.$$

Proposition 17. *The above algorithm correctly factorizes the polynomials of form (4) that satisfy assumption (*).*

Proof. Since $l_{in} = 1$ for all i we have $\lambda = f(0, \cdots, 0, 1) = g_j(0, \cdots, 0)$ for all $j = 2, \ldots, n-1$. Each g_j admits the factorization:

$$g_j(x_1, x_j) = \lambda . \prod_{i=1}^{k} (l_{i1}x_1 + l_{ij}x_j + 1)^{\alpha_j} \tag{5}$$

All these polynomials therefore have same factorization pattern as f (note in particular that the affine forms $l_{i1}x_1 + l_{ij}x_j + 1$ are nonconstant since $l_{i1} \neq 0$; and two of these forms cannot be proportional since the l_{i1} are distinct). It follows that the factorization of g_j discovered by the algorithm at step 2 is identical to (5) up to a permutation, i.e., we have $a_{ij} = l_{\sigma_j(i)1}$, $b_{ij} = l_{\sigma_j(i)j}$ and $\beta_{ij} = \alpha_{\sigma_j(i)}$ for some permutation $\sigma_j \in \mathfrak{S}_k$. Since the l_{i1} are distinct, after reordering at step 3 these $n-2$ permutations become identical, i.e., we have $a_i = l_{\sigma(i)1}$, $c_{ij} = l_{\sigma(i)j}$ and $\gamma_i = \alpha_{\sigma(i)}$ for some permutation σ. Finally, at step 4 the algorithm outputs the correct factorization $f(x) = \lambda . \prod_{i=1}^{k} l_{\sigma(i)}(x)^{\alpha_{\sigma(i)}}$. □

5.2 General case

In this section we present a black box algorithm that factors a homogeneous polynomial $f \in K[x_1, \ldots, x_n]$ of degree d into a product of d linear forms whenever this is possible, thereby lifting assumption (*) from Section 5.1. The algorithm is as follows.

1. Set $g(x) = f(A.x)$ where $A \in M_n(K)$ is a random matrix.

2. Attempt to factor g with the algorithm of Section 5.1. If this fails, reject f. In case of success, let $g'(x) = \lambda.l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$ be the factorization output by this algorithm.

3. Check that $f(x) = g'(A^{-1}.x)$ and output the corresponding factorization.

The random matrix at step 1 is constructed by drawing its entries independently at random from some large enough finite set $S \subseteq K$. The point of this random change of variables is that g will satisfy assumption (*) of Section 5.1 with high probability if f can be factored as a product of linear forms. The (quite standard) arguments needed to estimate the probability of success are presented in the proof of Theorem 18. Note also that by the Schwarz-Zippel lemma, A will be invertible with high probability.

At step 2 we need a black-box for g. Such a black box is easily obtained by composing the black box for f with the map $x \mapsto A.x$.

At step 3, we check the polynomial identity $f(x) = g'(A^{-1}.x)$ by evaluating the left and right-hand sides at one random point.

Theorem 18. *The above algorithm runs in polynomial time and determines whether f can be written as a product of linear forms. It outputs such a factorization if there is one.*

Proof. By the Schwarz-Zippel lemma, any factorization of f output at step 3 will be correct with high probability. So we only need to prove the converse: if f can be factored as a product of linear forms, the algorithm finds a correct factorization with high probability. Suppose therefore that

$$f(x) = L_1(x)^{\alpha_1} \cdots L_k(x)^{\alpha_k}$$

where no two linear forms L_i, L_j in this expression are proportional. Then $g(x) = f(A.x)$ can be written as

$$g(x) = \ell_1(x)^{\alpha_1} \cdots \ell_k(x)^{\alpha_k}$$

where $\ell_i(x) = L_i(A.x)$. If A is invertible, the linear forms in this expression will not be proportional. The coefficients of these linear forms are given by the expression:

$$\ell_{ij} = \sum_{p=1}^{n} L_{ip} A_{pj}. \tag{6}$$

If the entries A_{pj} of A are drawn from a set $S \subset K$, $\ell_{in} = 0$ with probability at most $1/|S|$ since $L_i \not\equiv 0$. These n coefficients will all be nonzero with probability at least $1 - n/|S|$; in this case we can factor out $\lambda = \prod_{i=1}^{k} \ell_{in}^{\alpha_i}$ to make sure that the coefficient of x_n in each linear form is equal to 1 as required by assumption (*). This gives the factorization

$$g(x) = \lambda.l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$$

where $l_i(x) = \ell_i(x)/\ell_{in}$. The same argument as for ℓ_{in} shows that ℓ_{i1} and therefore l_{i1} will be nonzero with high probability. To take care of assumption (*), it remains to check that the l_{i1} will be distinct with high probability. The condition $l_{i1} \neq l_{j1}$ is equivalent to $\ell_{i1}\ell_{jn} - \ell_{j1}\ell_{in} \neq 0$. By (6) this expression can be viewed as a quadratic form in the entries of A. From unique factorization and the hypothesis that the linear forms L_i, L_j are not proportional it follows that this quadratic form is not identically 0. We conclude again that it will be nonzero with high probability by the Schwarz-Zippel lemma.

We have established that $g(x) = f(A.x)$ satisfies (*) with high probability. In this case, by Proposition 17 the factorization of g at step 2 of the algorithm and the verification of the polynomial identity at step 3 will also succeed. □

6 Identifying the hyperplanes and their multiplicities

If a polynomial f can be factored as a product of linear forms, its zero set $Z(f)$ is a union of (homogeneous) hyperplanes. In this section we present an algorithm based on this simple geometric fact.

We can identify each hyperplane in $Z(f)$ by finding $n-1$ nonzero points that lie on it. Assume that f can be written as $f(x) = \lambda.l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$ where the linear forms l_i are not proportional. We will need a total of $k(n-1)$ points on $Z(f)$ to identify the k hyperplanes. Our algorithm begins with the determination of these $k(n-1)$ points.

1. Pick a random point $a \in K^n$ and $n-1$ random vectors $v_1, \ldots, v_{n-1}$ in K^n (or representatives of points in $\mathbb{P}(K^n)$ to be more precise).

   Let Δ_i be the line of direction v_i going through a. Compute the intersection $\Delta_i \cap Z(f)$ for $i = 1, \ldots, n-1$.

2. Output the $k(n-1)$ intersection points $a_1, \ldots, a_{k(n-1)}$ found at step 1.

In the sequel, we assume that $f(a) \neq 0$. This holds with high probability by the Schwarz-Zippel lemma.

At step 1 we compute $\Delta_i \cap Z(f)$ by finding the roots of the univariate polynomial $g(t) = f(a + tv_i)$. We obtain one point on each hyperplane $Z(l_1), \ldots, Z(l_k)$ except if v_i belongs to one of these hyperplanes. This can happen only with negligible probability. Moreover, these k points are distinct except if Δ_i goes through the intersection of two of these hyperplanes. Again, this happens with negligible probability (we explain in the proof of Theorem 19 how to obtain explicit bounds on the probabilities of these bad events). Since $a \notin Z(f)$, with high probability we find a total of $k(n-1)$ distinct points as claimed at step 2. Moreover, each hyperplane $Z(l_i)$ contains

exactly $n-1$ points. Note that at step 1 we have also determined k if this parameter was not already known in advance.

At the next stage of our algorithm we determine the k hyperplanes. We first determine the hyperplane going through a_1 as follows:

3. Find $n-2$ points $b_2, \ldots, b_{n-1}$ in the set $\{a_2, \ldots, a_{k(n-1)}\}$ such that each line $(a_1 b_j)$ is included in $Z(f)$.

4. Output the linear subspace $H_1 = \mathrm{Span}(a_1, b_2, \ldots, b_{n-1})$.

At step 3 we can find out whether a line $(a_1 a_j)$ is included in $Z(f)$ by checking that the univariate polynomial $g(t) = f(ta_1 + (1-t)a_j)$ is identically 0. This can be done deterministically with $k-1$ calls to the black box for f (indeed, if $g \not\equiv 0$ this polynomial has at most k roots, and we already know that $g(0) = g(1) = 0$). Alternatively, we can perform a single call to the black box by evaluating g at a random point.

Assume for instance that $Z(l_1)$ is the hyperplane going through a_1. In the analysis of the first two steps we saw that (with high probability) a_1 does not lie on any other $Z(l_j)$, and that exactly $n-2$ points $b_2, \ldots, b_{n-1}$ in $\{a_2, \ldots, a_{k(n-1)}\}$ lie on $Z(l_1)$. The algorithm identifies these points at step 3 (we will find exactly one point on each line Δ_i). It follows that the subspace H_1 output at step 4 is included in $Z(l_1)$. To conclude that $H_1 = Z(l_1)$, it remains to show that H_1 is of dimension $n-1$. Assume without loss of generality that $\{a_1\} = \Delta_1 \cap Z(l_1)$ and $\{b_j\} = \Delta_j \cap Z(l_1)$ for $j = 2, \ldots, n-1$. Then $a_1 = a + t_1 v_1$ and $b_j = a + t_j v_j$ for $j = 2, \ldots, n-1$. Here $v_1, \ldots, v_{n-1}$ are the directions chosen at step 1, and $t_1, \ldots, t_{n-1}$ are appropriate nonzero scalars. With high probability, the n vectors $a, v_1, \ldots, v_{n-1}$ are linearly independent. In this case, the family $a + t_1 v_1, \ldots, a + t_{n-1} v_{n-1}$ is of rank $n-1$ as desired.

The above analysis shows that steps 3 and 4 identify $H_1 = Z(l_1)$ with high probability. The $k-1$ remaining hyperplanes can be identified by repeating this procedure. For instance, to determine the second hyperplane H_2 we will remove the points $a_1, b_2, \ldots, b_{n-1}$ (which lie on H_1) from the set $\{a_1, \ldots, a_{k(n-1)}\}$ and we will determine the hyperplane going through the first of the $(k-1)(n-1)$ remaining points.

In the next stage of the algorithm we determine the multiplicities α_i of the linear forms l_i. This is done as follows:

5. Consider again the random point a and the random vector v_1 drawn at step 1. We have already computed the intersection points with $H_1 = Z(l_1), \ldots, H_k = Z(l_k)$ of the line Δ_1 of direction v_1 going through a. Recall that this was done by computing the roots $t_1, \ldots, t_k$ of the univariate polynomial $g(t) = f(a + tv_1)$.

   Let us assume without loss of generality that these roots are ordered so that $\{a + t_1 v_1\} = H_1 \cap \Delta_1, \ldots, \{a + t_k v_1\} = H_k \cap \Delta_1$. Now we

compute the multiplicities $\alpha_1, \ldots, \alpha_k$ of $t_1, \ldots, t_k$ as roots of g and we output these multiplicities.

If $f(x) = l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$, the multiplicities of the roots of g are indeed equal to $\alpha_1, \ldots, \alpha_k$ except if Δ_1 goes through the intersection of two of the hyperplanes $H_1, \ldots, H_k$. As already pointed out in the analysis of the first two steps, this happens only with negligible probability. Note that there is nothing special about Δ_1 at step 5: we could have used a new random line Δ instead.

The final stage of the algorithm is a normalization step.

6. At the beginning of this step we have determined linear forms l_i and multiplicities α_i so that $f(x) = \lambda . l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$ for some constant λ. We determine λ by one call to the black box for f at a point where the l_i do not vanish (for instance, at a random point).

We have obtained the following result.

Theorem 19. *Let $f \in K[x_1, \ldots, x_n]$ be a polynomial of degree d that admits a factorization $f(x) = \lambda . l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$ over $\overline{K}$, where no two linear forms l_i are proportional. The above algorithm determines such a factorization with high probability, and the number of calls to the black box for f is polynomial in n and d.*

Assume moreover that $K = \mathbb{Q}$ and that a factorization of f where $l_i \in \mathbb{Q}[x_1, \ldots, x_n]$ is possible. If the coefficients of these linear forms are of bit size at most s then all calls to the black box are made at rational points of bit size polynomial in n, d and s.

Proof. The correctness of the algorithm follows from the above analysis. Let us focus therefore on the case $K = \mathbb{Q}$ of the theorem. This result relies on a standard application of the Schwarz-Zippel lemma. More precisely, as explained in the analysis of the first two steps, we want to pick a random point a such that $f(a) \neq 0$ and random vectors $v_1, \ldots, v_{n-1}$ that do not belong to any of the hyperplanes. Moreover, the line Δ_i defined at step 1 should not go through the intersection of two hyperplanes. Let us pick the coordinates of a and of the v_i independently at random from a finite set $S \subseteq \mathbb{Q}$. By the Schwarz-Zippel lemma, $\Pr[f(a) = 0] \leq k/|S| \leq d/|S|$; and for any linear form l_j we have $\Pr[l_j(v_i) = 0] \leq 1/|S|$. As to Δ_i, let us bound for instance the probability of going through the intersection of the first two hyperplanes. Since Δ_i is the line of direction v_i going through a, it suffices to make sure that $l_2(a)l_1(v_i) - l_1(a)l_2(v_i) \neq 0$. By the Schwarz-Zippel lemma this happens with probability at least $1 - 2/|S|$.

Another constraint arising in the analysis of steps 3 and 4 is that $a, v_1, \ldots, v_{n-1}$ should be linearly independent. By the Schwarz-Zippel lemma, the corresponding determinant vanishes with probability at most $n/|S|$.

Note that the bounds obtained so far are independent of s. This parameter comes into play when we compute the intersections $\Delta_i \cap Z(f)$ at step 1. Recall that we do this by finding the roots of the univariate polynomial $g(t) = f(a + tv_i)$. The roots are:

$$t_1 = -l_1(a)/l_1(v_i), \ldots, t_k = -l_k(a)/l_k(v_i).$$

Then at step 3 we call the black box at points belonging to lines going through two of the $k(n-1)$ intersections points found at step 1. □

The algorithm presented in this section relies on a simple and appealing geometric picture, but it suffers from a drawback compared to the algorithms of sections 4 and 5:

Remark 20. *Assume that $K = \mathbb{Q}$. The above algorithm may need to call the black box for f at algebraic (non rational) points in the case where the linear forms l_i do not have rational coefficients. This is due to the fact that we call the black box at points that lie on the hyperplanes $l_i = 0$.*

By contrast, the algorithms of sections 4 and 5 always call the black box at integer points even when f has algebraic (non rational) coefficients. To see why this is true for the algorithm of Section 5, note that the main use of the black box is for performing bivariate interpolation. In Section 4, the black box is used only for the computation of the Lie algebra of f following Lemma 22 of [25]. More details on the black box calls performed by our three algorithms can be found in the appendix.

A Appendix: Cost of calls to the black box

In this section we compare the number of calls to the black box made by our three algorithms (for the cost of other operations, see Appendix B). A more thorough analysis would also take into account the size of points at which the black box is queried (for this, Remark 20 would become especially relevant).

It turns out that the hyperplane algorithm of Section 6 makes fewer calls to the black box than the other two. We also analyze these algorithms in the "white box" model, where we have access to an arithmetic circuit computing the input polynomial f. In that model, the cost of function evaluations becomes smallest for the Lie-theoretic algorithm of Section 4.

A.1 Lie-theoretic algorithm

In Section 4, the black box is used only for the computation of the Lie algebra of f. By Lemma 5, this boils down to the determination of linear dependence relations between the n^2 polynomials $x_j \frac{\partial f}{\partial x_i}$. The general problem of finding linear dependence relations between polynomials given by black box access is solved by the following lemma (see appendix A1 of [24] for a proof).

Lemma 21 (Lemma 14 in [25]). *Let $(f_1(x), f_2(x), \ldots, f_m(x))$ be an m-tuple of n-variate polynomials. Let $\mathcal{P} = \{a_i;\ 1 \leq i \leq m\}$ be a set of m points in K^n. Consider the $m \times m$ matrix*

$$M = (f_j(a_i))_{1 \leq i,j \leq m}.$$

With high probability over a random choice of $\mathcal{P}$, the nullspace of M consists precisely of all the vectors $(\alpha_1, \ldots, \alpha_m) \in K^m$ such that

$$\sum_{i=1}^{m} \alpha_i f_i(x) \equiv 0.$$

We therefore need to evaluate the n polynomials $\partial f/\partial x_i$ at n^2 random points. Note however that we have only access to a black box for f rather than for its partial derivatives. As is well known, it is easy to take care of this issue by polynomial interpolation. Suppose indeed that we wish to evaluate $\partial f/\partial x_i$ at a point a. Then we evaluate f at $d+1 = \deg(f)+1$ points on the line Δ which goes through a and is parallel to the i-th basis vector. From these $d+1$ values we can recover f and $\partial f/\partial x_i$ on Δ. We conclude that the Lie-theoretic algorithm performs $O(dn^3)$ calls to the black box for f.

These n^3 polynomial interpolations also have a cost in terms of arithmetic operations, but it is relatively small. Suppose indeed that we wish to compute $\partial f/\partial x_1$ at a point $a = (a_1, \ldots, a_n)$ with $a_1 \neq 0$. Consider the univariate polynomial $g(x) = f(a_1 x, a_2, \ldots, a_n)$. It suffices to compute $g'(1) = a_1 \partial f/\partial x_1(a)$. We can obtain $g'(1)$ as a fixed linear combination of $g(0), g(1), \ldots, g(d)$. One polynomial interpolation therefore requires one linear combination of values of f and one division (by a_1). We conclude for use in Appendix B.1 that the arithmetic cost of these n^3 interpolations is $O(n^3 d)$.

The Lie-theoretic algorithm admits an interesting optimization when the black box is implemented by an arithmetic circuit. Suppose indeed that we have access to an arithmetic circuit of size s computing f (this is the so-called "white box" model). The above analysis translates immediately into an arithmetic cost of order sdn^3 for the evaluation of the partial derivatives of f at our n^2 random points. But one can do much better thanks to the classical result by Baur and Strassen [1] (see also [30]), which shows that the n partial derivatives of an arithmetic circuit of size s can be evaluated by a single arithmetic circuit of size $O(s)$. This reduces the cost of evaluations from $O(sdn^3)$ to $O(sn^2)$. Moreover, the arithmetic cost of interpolations drops from $O(n^3 d)$ to 0 since we do not perform any interpolation in the white box model.

A.2 Bivariate projections

The algorithm of Section 5 recovers a factorization of the input polynomial f from the factorization of $n-2$ bivariate projections $g_1, \ldots, g_{n-2}$. The black

box is used only to obtain each g_j in dense form by interpolation.[12] This can be done deterministically by evaluating g_j on any set of the form $S \times S$ where $|S| = d+1$. We therefore need a total of $O(nd^2)$ function evaluations. Note that this is only $O(n^3)$ for $d = n$, i.e., smaller than the number of black box call performed by the Lie-theoretic algorithm. In general the bounds dn^3 and nd^2 obtained for our first two algorithms are not comparable since d could be much larger than n (this can happen when the exponents α_i in (1) are large enough).

There is no obvious improvement to this analysis of our second algorithm in the "white box" model described in Section A.1: the $O(nd^2)$ function evaluations translate into an arithmetic cost of order snd^2. Now the white box version of the Lie-theoretic algorithm becomes more interesting from the point of view of the cost of function evaluations: as explained above, this cost is only $O(sn^2)$. By contrast, this cost is $\Omega(sn^3)$ for the algorithm of Section 5 since $d \geq n$.

A.3 The hyperplane algorithm

In order to factor an input polynomial $f(x) = \lambda . l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$, this algorithm determines the hyperplanes $H_i = Z(l_i)$ together with their multiplicities α_i. The black box calls are performed at steps 1, 3 and 6 of the algorithm. We'll focus on steps 1 and 3 since Step 6 performs only one call to the black box.

At Step 1 we compute the intersection of $n-1$ lines $\Delta_1, \ldots, \Delta_{n-1}$ with $Z(f)$, the zero set of f. For this we need to interpolate f on each line; this requires a total of $(n-1)(d+1)$ calls to the black box.

The determination of a single hyperplane of $Z(f)$ is explained at step 3 of the algorithm, which we repeat here for convenience ($\{a_2, \ldots, a_{k(n-1)}\}$ are the intersection points found at Step 1):

3. Find $n-2$ points $b_2, \ldots, b_{n-1}$ in the set $\{a_2, \ldots, a_{k(n-1)}\}$ such that each line $(a_1 b_j)$ is included in $Z(f)$.

As explained in Section 6, the test $(a_1 b_j) \subseteq Z(f)$ can be implemented with one call to the black box at a random point on the line $(a_1 b_j)$. This test is repeated at most $k(n-1)$ times. We therefore need $O(kn)$ calls to determine a single hyperplane. There are k hyperplanes to determine, for a total cost of order $k^2 n$. We conclude that this algorithm makes $O(dn + k^2 n)$ calls to the black box. The two terms dn and $k^2 n$ in this bound are in general incomparable since $d \geq k$ is the only relation between $d = \deg(f)$ and the number k of distinct factors.

[12] There is also an additional call to the black box for verification of the final result, see Step 3 in Section 5.2.

In order to compare with the Lie-theoretic algorithm we whould set $k = n$ since that algorithm applies only in this situation. The cost of the hyperplane algorithm becomes $O(dn + n^3)$; this is smaller than the $O(dn^3)$ bound obtained for the Lie-theoretic algorithm. Note however that the latter algorithm becomes cheaper in the white box model: as explained in Section A.1 the arithmetic cost of function evaluations is only $O(sn^2)$ when f is given by an arithmetic circuit of size s. This should be compared to a cost of order $s(dn+n^3)$ for the hyperplane algorithm (like the bivariate algorithm, it does not seem to admit any interesting optimization in the white box model).

Finally, the hyperplane algorithm should be compared to bivariate projections. In number of calls to the black box, the latter algorithm is always as expensive or more expensive than the former (compare $dn + k^2n$ to d^2n).

B Appendix: Cost of other operations

In this section we continue the analysis of our three algorithms. Appendix A dealt with the number of calls to the black box. Here we estimate the cost of "other operations", which consist mostly of:

- arithmetic operations, in K or in an extension of K.
- certain non-algebraic steps such as eigenvalue computations or the factorization of univariate polynomials.

The bounds that we give should only be viewed as very rough estimates of the algorithms' complexity since we do not perform a full analysis at the level of bit operations.[13]

B.1 Lie-theoretic algorithm

In this section we analyze more precisely the algorithm of Section 4.2. We'll focus first on the complexity of deciding the existence of a suitable factorization over $\overline{K}$. This is done in the first three steps of the algorithm. Note that the corresponding steps for the case of equal exponents (Section 4.1) only differ by the presence of $n-1$ trace computations. The cost of trace computations turns out to be negligible, so this analysis applies to the two versions of our algorithm.

At Step 1 of the algorithm we compute a basis of the Lie algebra of f. The Lie algebra is the nullspace of a certain matrix M of size $m = n^2$ which we have already computed as explained in Appendix A.1. A basis of the nullspace can be computed with $O(m^3)$ arithmetic operations by Gaussian elimination, and with $O(m^\theta)$ operations using fast linear algebra [4].

[13] Note that a complexity analysis at the level of bit operations is also omitted from the paper by Kalftofen and Trager [21] on black box factorization.

Here θ denotes any exponent strictly larger than ω, the exponent of matrix multiplication.

At Step 3 we first check that the matrices $B_1, \ldots, B_{n-1}$ commute, where $B_1, \ldots, B_{n-1}$ is the basis of the Lie algebra found at Step 1. This can be done in $O(n^{2+\omega})$ arithmetic operations. This is negligible compared to the cost $O(n^{2\theta})$ of the first step since $\theta > \omega \geq 2$.

Then we check that the B_i are all diagonalizable. Recall from Section 2.1 that B_i is diagonalizable over $\overline{K}$ iff its minimal polynomial m_i has only simple roots. The minimal polynomial can be computed in $O(n^\theta)$ arithmetic operations [20, 37]. Then we need to check that $\gcd(m_i, m_i') = 1$. The cost of computing the gcd is negligible compared to n^θ. The cost of the $n-1$ diagonalizability tests is $O(n^{1+\theta})$, which is again negligible compared to Step 1. We conclude that the existence of a suitable factorization of f can be decided in $O(n^{2\theta})$ arithmetic operations.

At Step 4 we perform a simultaneous diagonalization of the B_i. As suggested in Section 2.2 and in Section 4, this can be done by diagonalizing a random combination R of the B_i's. For this, as recalled in Section 2.1 we can first compute the eigenvalues $\lambda_1, \ldots, \lambda_n$ of R (a non algebraic step). Then we compute a basis of $\ker(R - \lambda_i I)$ for all i. One basis can computed in $O(n^\theta)$ arithmetic operations, so $O(n^{1+\theta})$ is a rough estimate on the number of arithmetic operations needed to compute a transition matrix T (we will not try to improve it since it is dominated by the cost of Step 1). Note that these arithmetic operations take place in $K[\lambda_1, \ldots, \lambda_n]$, so counting such an operation as "one step" is probably most appropriate when the λ_i lie in K, or when we work with approximations of the λ_i.

Once T is known, the $n-1$ diagonal matrices $D_i = T^{-1} B_i T$ can be computed at a cost of $O(n^{1+\theta})$ arithmetic operations. Then, as explained at Step 5, we obtain the exponents α_i as the orthogonal of the space spanned by the D_i in the space of diagonal matrices. Alternatively, the α_i can be obtained without knowledge of T as explained after Proposition 16: the n exponents are the eigenvalues of matrix H which is obtained as the unique solution of a system of $\Theta(n^3)$ equations in n^2 unknowns. This approach looks rather expensive since solving a *square* system in n^2 unknowns would already take $O(n^{2\theta})$ arithmetic operations.

The above analysis can be summarized as follows.

Proposition 22. *The algorithm of Section 4.2 decides in $O(n^{2\theta} + n^3 d)$ arithmetic operations whether the input polynomial f admits a factorization of the form (1) over $\overline{K}$. If such a factorization exists, it can be computed within the same number of arithmetic operations and the additional computation of the eigenvalues of a matrix $R \in M_n(K)$.*

In the white box model of Appendix A.1, the number of arithmetic operations drops from $O(n^{2\theta} + n^3 d)$ to $O(n^{2\theta})$.

The term $n^3 d$ in Proposition 22 is due to the arithmetic cost of inter-

polations as explained in Appendix A.1. Towards a more thorough analysis of this algorithm one could attempt to estimate its bit complexity, assuming for instance for simplicity that f admits a factorization with the l_i in $\mathbb{Z}[x_1, \ldots, x_n]$.

B.2 Bivariate projections

The algorithm from Section 5 recovers a factorization of f from $n-2$ bivariate factorization. A state of the art algorithm for the latter task can be found in [28], where the following reduction from bivariate to univariate factorization is provided.

Theorem 23. *Let $\mathbb{K}$ be a field of characteristic 0, and $F \in \mathbb{K}[x, y]$ a bivariate polynomial of degree d_x in the variable x and d_y in the variable y. There is a probabilistic algorithm that factors F in $O((d_x d_y)^{1.5})$ arithmetic operations. Moreover, the algorithm performs irreducible factorizations of polynomials in $\mathbb{K}[y]$ whose degree sum is at most $d_x + d_y$.*

We have omitted the cost of generating random field elements from the statement of the theorem. A deterministic version of this result is also provided in [28], with a slightly higher arithmetic cost: $O((d_x d_y)^{(\theta+1)/2})$ instead of $O((d_x d_y)^{1.5})$. The univariate factorizations in Theorem 23 can be viewed as an analogue of the eigenvalue computations in Proposition 22.

For an input polynomial f with coefficients in K, it may be the case that a factorization into products of linear forms exists only in an extension $\mathbb{K}$ of K. We will therefore need to apply Theorem 23 to such a field extension, and the arithmetic operations in Theorem 23 will also take place in this field extension. We have already made a similar remark for the algorithm of Section 4 in Section B.1.

If the input f has degree d, we can take $d_x = d_y = d$ and we conclude that the algorithm of Section 5 will make $O(nd^3)$ arithmetic operations. For $d = n$ this is smaller than the arithmetic cost $O(n^{2\theta})$ in Proposition 22, but the latter bound becomes smaller if d significantly larger than n.

A complete analysis should also take the cost of univariate factorizations into account. Assume for instance that $K = \mathbb{K} = \mathbb{Q}$. A polynomial time algorithm for this task was first by given Lenstra, Lenstra and Lovasz [29]. This remains a relatively expensive task despite several improvements (see [3] for an exposition and more references). However, we only need to find the linear factors of F (together with their multiplicities). This boils down to finding the rational roots of a univariate polynomial, a task which (as already pointed out in Section 2.1) has an essential quadratic binary cost ([3], Proposition 21.22).

As an alternative to Theorem 23 one may use the absolute factorization algorithm by Chèze and Lecerf [12]. This algorithm only performs arithmetic operations (no univariate factorization is involved). Moreover, the number

of arithmetic operations is barely higher: $\tilde{O}(d^3)$ instead of $O(d^3)$, where the $\tilde{O}$ notation hides logarithmic terms. We refer to [12] for a more precise statement of their result and a description of the output representation. One advantage of their algorithm is that all arithmetic operations take place in the coefficient field K of the input polynomial, even if the factors only exist in a field extension $\mathbb{K}$. Finally, we note that their algorithm only applies to squarefree bivariate polynomials. For a reduction from the general case to the squarefree case we refer to Section 4.2 of [28].

So far, we have not addressed the arithmetic cost of converting from black box representation to dense bivariate representation. As pointed out in Section A.2, this can be done by interpolating each of the $n-2$ bivariate polynomials on a set of size $(d+1)^2$. There are several ways of doing this at negligible cost compared to dense bivariate factorization.

First, recall that a univariate polynomial of degree d can be interpolated from its values at roots of unity in $O(d \log d)$ arithmetic operations using the Fast Fourier Transform. The cost of univariate interpolation at an arbitrary set of $d+1$ points is a little higher but remains $\tilde{O}(d)$, see Section 10 of [40] for details.

Returning to bivariate polynomials, one option is to use the two-dimensional FFT. Its cost remains $O(N \log N)$, where N is the number of interpolation points. Here, $N = (d+1)^2$ and we interpolate on $S \times S$ where S is the set of $(1+d)$-th roots of unity. Another option is to perform the Kronecker substitution $y = x^{d+1}$ and interpolate the polynomial $g(x) = f(x, x^{d+1})$ using one of the aforementioned univariate methods. The coefficients of f can be recovered uniquely from those of g.

B.3 The hyperplane algorithm

Recall that the algorithm proposed in Section 6 factors an input polynomial $f(x) = \lambda . l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$ from $n-1$ univariate polynomial factorizations. Each univariate polynomial is of the form $g(t) = f(a + tv_i)$ and its coefficients must be determined by interpolation at Step 1. As already mentioned in Section B.2, this can be done in $O(d \log d)$ arithmetic operations by FFT from $d+1$ values of g. In order to obtain one value of g we must compute the coordinates of $a + tv_i$ before calling the black box for f, at a cost of n arithmetic operations. Interpolating g therefore takes $O(d \log d + dn)$ arithmetic operations. Since we have $n-1$ such polynomials to interpolate, the arithmtic cost of interpolations is $O(dn(n + \log d))$.

The roots $a_1, \ldots, a_{k(n-1)}$ of the $n-1$ univariate polynomials are used at steps 3 and 4 to determine the zero sets of the l_i. At step 5, the multiplicities of the roots of the first polynomial yield the exponents α_i.

At step 3 we test whether the line $(a_1 b)$ is included in $Z(f)$, where b is one the $k(n-1)$ roots. This is done by evaluating the black box at a random linear combination $ta_1 + (1-t)b$. The coordinates of this point

can be computed in $O(n)$ arithmetic operations. Repeating this for all the $k(n-1)$ roots takes $O(kn^2)$ operations.

At step 4 we determine $H_1 = \mathrm{Span}(a_1, b_2, \ldots, b_{n-1})$, where the b_i have been found at Step 3. Finding an equation for H_1 amounts to solving a linear system, and can be done in $O(n^\theta)$ arithmetic operations as recalled in Appendix B.1. The combined cost of the determination of H_1 at steps 3 and 4 is therefore $O(kn^2 + n^\theta)$. This is repeated for all the hyperplanes, at a total cost of $O(k(kn^2 + n^\theta))$ operations. We recall that these arithmetic operations may take place in a field extension.

Finally, at Step 6 we evaluate the product $\prod_{i=1}^{k} l_i(x)^{\alpha_i}$ at some point x and divide $f(x)$ by this product to determine the normalization factor λ. If we use repeated squaring to evaluate the powers $l_i(x)^{\alpha_i}$, we can complete Step 6 in $O(k(n+\log d))$ arithmetic operations. Since $k \leq d$ this is negligible compared to the cost of the univariate interpolations at Step 1. The above analysis can be therefore be summarized as follows.

Proposition 24. *The algorithm of Section 6 obtains a factorization of the form $f(x) = \lambda . l_1(x)^{\alpha_1} \cdots l_k(x)^{\alpha_k}$ using $O(k(kn^2+n^\theta)+dn(n+\log d))$ arithmetic operations. The algorithm also needs to compute the roots of $n-1$ univariate polynomials of degree d, and for one of these polynomials it needs to determine the multiplicities of roots.*

For comparison with the Lie-theoretic algorithm, setting $k = n$ in Proposition 24 yields a count of $O(n^4 + dn(n + \log d))$ arithmetic operations. If d remains polynomially bounded in n, this is always smaller than the corresponding $O(n^{2\theta} + n^3 d)$ bound for the black box version of the Lie-theoretic algorithm.[14] In the white box model, the arithmetic cost of that algorithm drops to $O(n^{2\theta})$. The Lie-theoretic algorithm therefore becomes preferable from the point of view of the arithmetic cost when d exceeds $n^{2\theta-2}$.

Acknowledgements

P.K. would like to thank Gilles Villard for useful pointers to the literature on computational linear algebra.

References

[1] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, 1983.

[2] Åke Björck. *Numerical methods in matrix computations*. Springer, 2016.

[14] We recall that the term $n^{2\theta}$ comes from the computation of the Lie algebra of f, and the term $n^3 d$ from the arithmetic cost of polynomial interpolation. The arithmetic computations for these two tasks take place in the coefficient field K of f rather than in a field extension.

[3] Alin Bostan, Frédéric Chyzak, Marc Giusti, Romain Lebreton, Grégoire Lecerf, Bruno Salvy, and Éric Schost. *Algorithmes efficaces en calcul formel*. Published by the authors, 2017.

[4] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.

[5] P Bürgisser and F Cucker. *Condition: The geometry of numerical algorithms*, volume 349 of *Grundlehren der Mathematischen Wissenschaften*. Springer Verlag, 2013.

[6] Peter Bürgisser, Ankit Garg, Rafael Oliveira, Michael Walter, and Avi Wigderson. Alternating minimization, scaling algorithms, and the null-cone problem from invariant theory. In *Innovations in Theoretical Computer Science (ITCS)*, 2018.

[7] Peter Bürgisser and Christian Ikenmeyer. Deciding positivity of Littlewood–Richardson coefficients. *SIAM Journal on Discrete Mathematics*, 27(4):1639–1681, 2013.

[8] Peter Bürgisser, Christian Ikenmeyer, and Greta Panova. No occurrence obstructions in geometric complexity theory. In *Proc. 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 386–395, 2016.

[9] Peter Bürgisser, Felipe Cucker, and Elisa Rocha Cardozo. On the condition of the zeros of characteristic polynomials. *Journal of Complexity*, 42:72 – 84, 2017.

[10] Enrico Carlini. Reducing the number of variables of a polynomial. In *Algebraic geometry and geometric modeling*, Math. Vis., pages 237–247. Springer, Berlin, 2006.

[11] Enrico Carlini, Maria Virginia Catalisano, and Anthony V Geramita. The solution to the Waring problem for monomials and the sum of coprime monomials. *Journal of Algebra*, 370:5–14, 2012.

[12] Guillaume Chèze and Grégoire Lecerf. Lifting and recombination techniques for absolute factorization. *Journal of Complexity*, 23(3):380–420, 2007.

[13] Pierre Comon and Giorgio Ottaviani. On the typical rank of real binary forms. *Linear and multilinear algebra*, 60(6):657–667, 2012.

[14] David Cox, John Little, and Donal O'shea. *Using algebraic geometry*, volume 185 of *Graduate Texts in Mathematics*. Springer, 2006.

[15] Richard DeMillo and Richard Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1977.

[16] Shuhong Gao. Factoring multivariate polynomials via partial differential equations. *Mathematics of computation*, 72(242):801–822, 2003.

[17] Ignacio García-Marco, Pascal Koiran, and Timothée Pecatte. Polynomial equivalence problems for sums of affine powers. To appear in Proc. ISSAC 2018.

[18] Ankit Garg, Leonid Gurvits, Rafael Oliveira, and Avi Wigderson. A deterministic polynomial time algorithm for non-commutative rational identity testing. In *Proc. 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 109–117, 2016.

[19] Mark Giesbrecht. Fast algorithms for rational forms of integer matrices. In *Proceedings of the international symposium on Symbolic and algebraic computation*, pages 305–311. ACM, 1994.

[20] Mark Giesbrecht. Nearly optimal algorithms for canonical matrix forms. *SIAM Journal on Computing*, 24(5):948–969, 1995.

[21] E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, 9(3):301–320, 1990.

[22] Erich Kaltofen. Polynomial-time reductions from multivariate to bi-and univariate integral polynomial factorization. *SIAM Journal on Computing*, 14(2):469–489, 1985.

[23] Erich Kaltofen. Factorization of polynomials given by straight-line programs. In *Randomness and Computation*, pages 375–412. JAI Press, 1989.

[24] Neeraj Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, January 2011.

[25] Neeraj Kayal. Affine projections of polynomials. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 643–662, 2012.

[26] Hidetsune Kobayashi, Tetsuro Fujise, and Akio Furukawa. Solving systems of algebraic equations by a general elimination method. *Journal of Symbolic Computation*, 5:303–320, 1988.

[27] Joseph M. Landsberg. *Geometry and complexity theory*, volume 169 of *Studies in Advanced Mathematics*. Cambridge University Press, 2017.

[28] Grégoire Lecerf. New recombination algorithms for bivariate polynomial factorization based on Hensel lifting. *Applicable Algebra in Engineering, Communication and Computing*, 21(2):151–176, 2010.

[29] Arjen Lenstra, Hendrik Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[30] Jacques Morgenstern. How to compute fast a function and all its derivatives: A variation on the theorem of Baur-Strassen. *ACM SIGACT News*, 16(4):60–62, 1985.

[31] Ketan Mulmuley, Hariharan Narayanan, and Milind Sohoni. Geometric complexity theory III: on deciding nonvanishing of a Littlewood–Richardson coefficient. *Journal of Algebraic Combinatorics*, 36(1):103–110, 2012.

[32] Ketan Mulmuley and Milind Sohoni. Geometric complexity theory I: An approach to the P vs. NP and related problems. *SIAM Journal on Computing*, 31(2):496–526, 2001.

[33] Ketan Mulmuley and Milind Sohoni. Geometric complexity theory II: Towards explicit obstructions for embeddings among class varieties. *SIAM Journal on Computing*, 38(3):1175–1206, 2008.

[34] Clément Pernet and Arne Storjohann. Faster algorithms for the characteristic polynomial. In *Proceedings of the 2007 international symposium on Symbolic and algebraic computation*, pages 307–314. ACM, 2007.

[35] J. T. Schwarz. Fast probabilistic algorithms for verification of polynomials identities. *Journal of the ACM*, 27:701–717, 1980.

[36] Michael F Singer and Felix Ulmer. Linear differential equations and products of linear forms. *Journal of Pure and Applied Algebra*, 117:549–563, 1997.

[37] Arne Storjohann. Deterministic computation of the Frobenius form. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 368–377, 2001.

[38] Mark van Hoeij, Jean-François Ragot, Felix Ulmer, and Jacques-Arthur Weil. Liouvillian solutions of linear differential equations of order three and higher. *Journal of Symbolic Computation*, 28(4-5):589–609, 1999.

[39] Gilles Villard. Fast parallel algorithms for matrix reduction to normal forms. *Applicable Algebra in Engineering, Communication and Computing*, 8(6):511–537, 1997.

[40] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University press (third edition), 2013.

[41] J.H. Wilkinson. The perfidious polynomial. In G.H. Golub, editor, *Studies in numerical analysis*, pages 1–28. American Mathematical Society, 1984.

[42] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and algebraic computation*, pages 216–226. Springer, 1979.